732A92 - Text Mining Project Report

Detecting Sarcasm on News Headlines

Yiran Wang
yirwa560

LINKÖPINGS UNIVERSITET

Abstract

Online resources, for example, news, become overwhelming to us due to the rising importance of the internet in our daily lives. It leads to more and more online news providers using sarcastic headlines to draw people's attention. Headlines expressed in sarcastic ways can be misunderstood and cause trouble when the wrong facts get spread. Thus, detecting sarcasm automatically is in need now more than ever. In this project, LSTM, BERT, and Naive Bayes (baseline) models are built to identify sarcasm in news headlines using a labeled news headline dataset. All models are fine-tuned using grid-search and holdout/cross-validation to obtain the best sets of hyper-parameters. In addition, a hybrid neural network model from a related article is also reproduced and fine-tuned on the same dataset for comparison. As a result, both LSTM and BERT models show better performance, in terms of F1 score and accuracy, than the baseline model. The LSTM and the hybrid models are faster to train than the BERT model. The BERT model is easier to construct and takes fewer epochs to converge due to the advantage of transfer learning. BERT also outperforms LSTM and the hybrid model, with 86% accuracy on the test dataset.

Contents

List of Figures

List of Tables

1. Introduction

Nowadays social media has shown a great impact on our daily life, among which, news website has become one of the main sources that people learn the world from. As a booming industry, there is inevitable competition between news providers. Many editors then start to catch readers' attention by using sarcastic or ironic headlines for their news articles. Sarcasm usually means to express itself in the opposite way (Barbieri et al., 2014). However, it is not an easy task even for a human to identify it, and it can be a problem if people get and spread the wrong idea about the news with sarcastic headlines.

As a result, a machine learning model that can detect sarcasm automatically and effectively may provide great help. Traditional machine learning methods that rely on primitive level features are not enough to identify whether a sentence is sarcastic or not. Advanced models that can capture not only the meaning of words but also the relationship between words are required for this task. Thus, in this project, different models are built and trained to address this intention.

With the labeled dataset of news headlines, a pre-processing procedure is done by removing stop words and non-alphabetic words. The data is also analyzed to reveal potential problems. Then it is shuffled and randomly split into training, validation, and test datasets. A simple model of Naive Bayes classifier using Count Vectorizer is constructed as the baseline for this task. The LSTM model using pre-trained GloVe word embedding and pre-trained BERT model with its tokenizer are built and compared as advanced models. For each model, the best setup of hyper-parameters is gained by fine-tuning using grid-search and holdout/cross-validation approach. A hybrid model with LSTM-CNN architecture proposed by Mehndiratta and Soni (2019) for a similar task is also reproduced, fine-tuned on the same dataset using in this project, and compared.

Both LSTM and BERT models give better performance than the Naive Bayes baseline model in terms of accuracy and F1 score. The BERT model outperforms the LSTM and the hybrid model.

2. Theory

2.1 Count Vectorizer

When dealing with text data, it needs to be converted to numerical data for putting into machine learning models. Count Vectorizer is a simple way to do this, by mapping each unique word in the dataset to a vector that represents the number of appearances this word occurs in every document. The higher the count value is, the more frequent the word is in the document.

2.2 Tf-idf Vectorizer

Similarly, Tf-idf is another way to vectorize text data using Tf-idf values instead of word count. Tf-idf value is the product of the term frequency (the number of occurrences of the word in a document) and inverse document frequency (a measure of how rare or common the word is across all documents). The higher the Tf-idf value is, the more important the word relates to the document.

2.3 Naive Bayes

Naive Bayes classifiers are a collection of probabilistic classifiers which are based on Bayes' Theorem with an assumption that each variable is independent and equally attributed to the output. (Hastie, 2001) The probability model is showing as equation 2.1.

$$\hat{y} = argmax_k P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{2.1}$$

In this project, the Multinomial Naive Bayes (equation 2.2) (McCallum & Nigam, 1998) where each observation follows a multinomial distribution is used on the text classification, with observation representing the frequency of a word in documents.

$$\hat{y} = \frac{(\sum_{i=1}^{n} x_i)!}{\prod_{i=1}^{n} x_i!} \prod_{i=1}^{n} p_{ki}^{x_i} \tag{2.2}$$

2.4 GloVe

Word embedding is another way to represent words in a numeric way for the models to learn. Unlike vectorizer, a word embedding is to represent a vocabulary of words in a d-dimensional vector space where semantic similar words will be placed closer to each other. It is denser and lower-dimensional, which creates a computational benefit (Goldberg, 2017). It can be learned by either along with the neural network model on a certain task, or a more unsupervised learning approach by using document statistics. Learning an own word embedding usually requires a large amount of training data and computational resources, but it can

give a more topic-specific embedding. Due to the limitation of the training data in this project, and the generalization of the topic of the news headlines, using a pre-trained word embedding is a better approach.

GloVe is one of the most popular pre-trained word embeddings. It is a distributed word representation model developed by Pennington et al.(2014) at Stanford using an unsupervised learning algorithm in 2014. Classic ways of learning word vectors include using global matrix factorization techniques such as LSA(Latent semantic analysis)(Deerwester et al., 1990) which uses global statistics but is not good at capturing meaning or demonstrating analogies. Another classic way is to use local context window approaches such as the skip-gram model(Mikolov et al., 2013) which learns by predicting the surrounding words given a certain word. The Skip-gram model is good at dealing with analogy tasks, but it doesn't utilize global statistics since they train only on local windows of neighboring words. GloVe is a way to utilize the main benefit of both these two classic methods, to be more specific, it is based on a co-occurrence matrix of words and learns to encode word vectors that include the probability ratio. As a result, the word vectors generated by GloVe outperform those generated from other models, for example, Word2Vec, from the same context in many ways including word analogy and similarity tasks.

2.5 LSTM

RNN (recurrent neural networks) is generally good at processing sequence data but it can only carry information for a short term due to its vanishing gradient problem during the backpropagation phase. LSTM (Long short-term memory) is created to solve this. It is a special recurrent neural network architecture using an appropriate gradient-based learning algorithm that can process sequences of data and track long-term dependencies with vanishing gradient problem occurrence that happens very often in classic recurrent neural networks greatly reduced. (Hochreiter & Schmidhuber, 1997)

LSTM is usually formed with a set of units including memory cell, input and output gate, and forget gate. Memory cells are designed to make LSTM be able to carry the information through long time lags. Forget gate with a sigmoid layer can regulate the passing information to filter only the important part and discard the irrelevant information. Input gate with a sigmoid and a tanh layer is to update the state of the cell, for example adding necessary information from the current time step. And output gate which includes a sigmoid function on the cell state, a tanh process of the cell state, and a multiplication of two results, is to output for the next state. The base structure can be seen in figure 2.1 (Hochreiter & Schmidhuber, 1997).

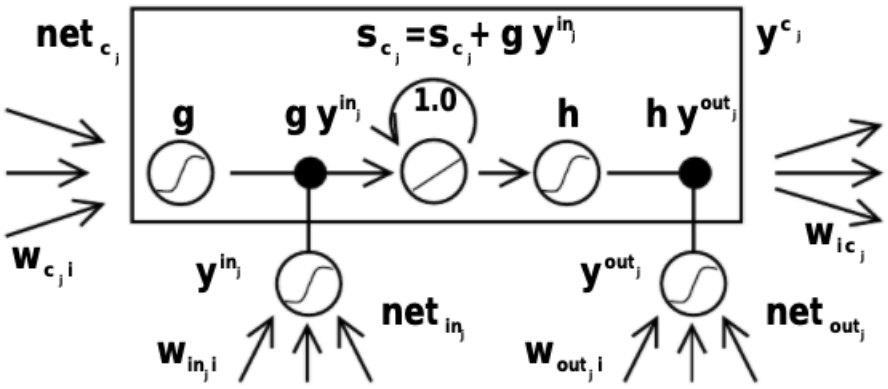

Figure 2.1: Architecture of LSTM module. The box in the middle represents memory cell. G is its forget gate. In and out are the input and output gates respectively.

2.6 BERT

BERT(Bidirectional Encoder Representations from Transformers) is a machine learning model which is based on transformers and can be fine-tuned on different neural language process tasks. BERT was developed by Jacob Devlin and his colleagues from Google in 2018. (Devlin et al., 2018)

Transformer (developed in 2017 by Google Brain(Vaswani et al., 2017)) is a machine learning model which is designed to process sequential data. Unlike RNN, it can deal with a word in any location instead of having to proceed from the beginning or end in order, with the help of the self-attention mechanism. In addition to the base encoder-decoder structure which is widely used in models in sequential tasks, Transformer uses self-attention (figure 2.2(Vaswani et al., 2017)).

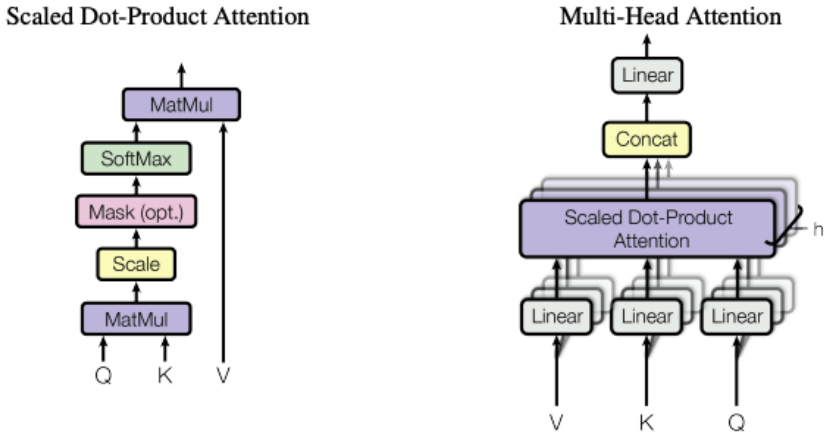

Figure 2.2: Architecture of single head and multi head attention

Transformer is the core of BERT with using a variable number of self-attention heads and encoding layers (figure 2.3 (Vaswani et al., 2017)).

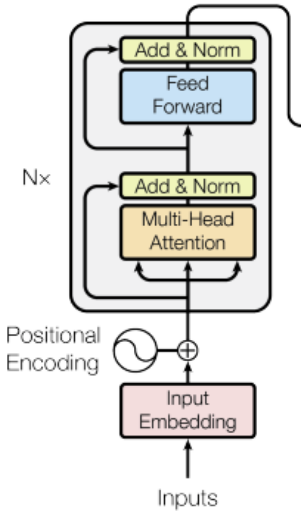

Figure 2.3: Architecture of encoding layer

One important capability of BERT is that its transformer encoder is designed to read the entire sequence at one time, instead of reading it either from left to right or from right to left like standard directional language models. This attribute allows the model to consider all the surrounding words when learning the context of a specific word.

BERT is trained using two strategies: MLM(Masked LM) and NSP(Next Sentence Prediction). MLM training is having some words masked and unseen and training the model to predict the masked words based on the surroundings. This method allows each word to have a meaning which is defined by its context. While traditional word embeddings, such as GloVe, map each word to a fixed vector that is pre-defined and only represents one aspect of its meaning. NSP training is to predict the relationship between every two sentences, for example, if they are logically connected.

There are two phases in BERT, pre-training and fine-tuning as figure 2.4 (Devlin et al., 2018) shows. During the pre-training phase, it uses document-level corpus including English Wikipedia text and the BooksCorpus (Zhu et al., 2015) to do the two unsupervised tasks which are MLM and NSP as described previously. During fine-tuning section, BERT just fine-tunes all the parameters from end-to-end by putting our task-specific inputs and outputs, with initializing with the pre-trained parameters. The Transformer (self-attention technique) makes this phase easier to adapt to different types of tasks. Although the number of trainable parameters looks huge, it is much faster than the pre-training phase. In fact in this project, only a minimal number of them will be learned from scratch, because only one additional layer is added along with loaded BERT model in this task-specific model. The fine-tuning procedure for the task in this project can be shown as figure 2.5 (Devlin et al., 2018).

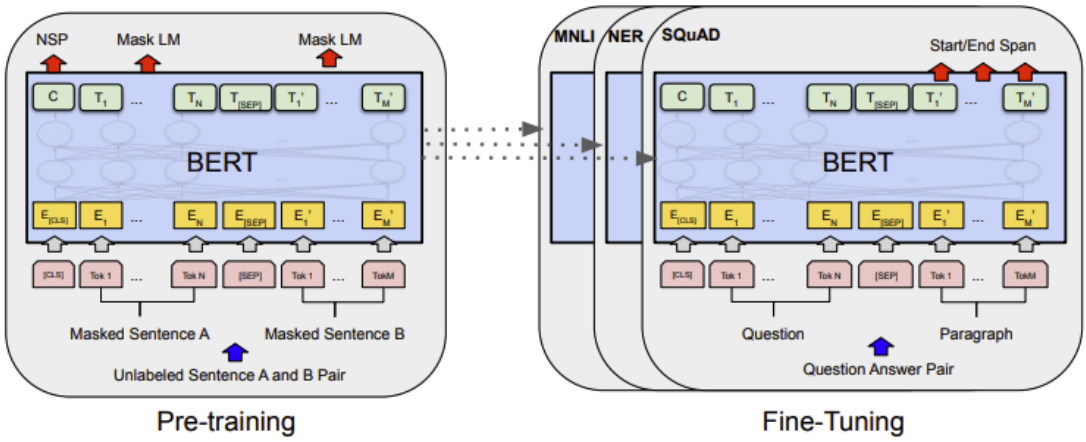

Figure 2.4: Architecture of BERT. Pre-training and fine-tuning phases.

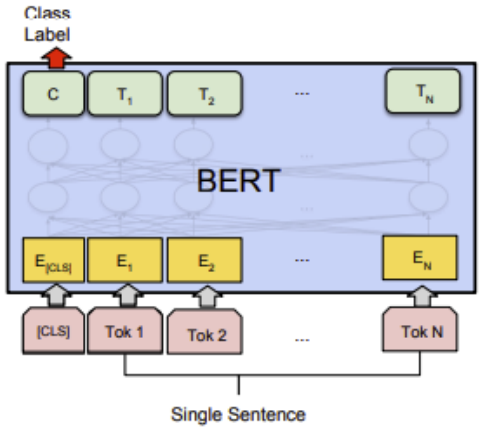

Figure 2.5: Architecture of fine-tuning procedure of BERT, associate with the task in this project.

There are two main architectures: BERT BASE and BERT LARGE. They both have the core technique (Transformer with bidirectional self-attention). The LARGE model is much bigger with 240 million parameters (the BASE model has around 110 million parameters). Devlin et al.(2018) describe the LARGE model appears to have a generally better performance than the BASE model, but the BASE model still outperforms other classic models such as OpenAI GPT (in which the Transformer is unidirectional) in many cases. In this project, due to limited memory access, only the BASE model is experimented with.

3. Data

The data this is used in this project is called "News Headlines Dataset For Sarcasm Detection"(Misra, 2021), consists of three variables:

- **is_sarcastic**: label whether the observation is sarcastic or not.
- **headline**: the headline of the news.
- **article_link**: the link to the original news article.

The data is mainly collected from two news websites: TheOnion[1] which produces a sarcastic version of real news with sarcastic headlines and HuffPost[2] where there are real and non-sarcastic news headlines. The dataset has a relatively good quality, on the one hand, the labels are reliable due to the two distinct sources. On the other hand, the headlines are collected from news websites, which indicates that they are generally written by professionals for a formal purpose. Furthermore, the news articles widely cover many hot topics and events. Thus it is not topic-based, which will be further proved by later data analyzing. This makes it possible to use pre-trained word embedding vectors directly without having to train on our own or even further tuning.

At the first glance of the data, the headlines are already in lowercase and labels are clear and intact. An example sample of the actual data is shown in table 3.1.

Table 3.1: News headlines Dataset snippet

is_sarcastic	headline	article_link
1	thirtysomething scientists unveil doomsday clock of hair loss	https://www.theonion.com/thirtysomething-scientists-unveil-doomsday-clock-of-hai-1819586205
0	dem rep. totally nails why congress is falling short on gender, racial equality	https://www.huffingtonpost.com/entry/donna-edwards-inequality_us_57455f7fe4b055bb1170b207
0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-veggies-9-delici_b_8899742.html
1	inclement weather prevents liar from getting to work	https://local.theonion.com/inclement-weather-prevents-liar-from-getting-to-work-1819576031
1	mother comes pretty close to using word 'streaming' correctly	https://www.theonion.com/mother-comes-pretty-close-to-using-word-streaming-cor-1819575546

Before further analyzing the data, a pre-process is done by removing the stop words and non-alphabetic words using Spacy. Then duplicates and empty strings are removed. Table 3.2 shows an example of pre-processing results.

Table 3.2: Data pre-process before and after comparison

Before Pre-process	After Pre-process	is_sarcastic
scientists receive $10 million grant to melt stuff	scientists receive million grant melt stuff	1
listen, area boss gets it	listen area boss gets	1
pope francis finds self in hell after taking wrong turn in vatican catacombs	pope francis finds self hell taking wrong turn vatican catacombs	1
report finds poor often hit hardest by 18-wheelers	report finds poor hit hardest	1
impatient nation demands supreme court just get to the gay stuff	impatient nation demands supreme court gay stuff	1
10 lessons we learned from diane keaton's new book	lessons learned diane keaton new book	0
brazilian artists pay tribute to olympic refugee team in stunning murals	brazilian artists pay tribute olympic refugee team stunning murals	0
hiring your first employee - what you don't know can hurt your business	hiring employee know hurt business	0
watch dirt bikers get a bit too close to nature	watch dirt bikers bit close nature	0

[1]https://www.theonion.com/
[2]https://www.huffpost.com/

After pre-processing, the total amount of valid data is 28464, and the class distribution is quite even, with 48% being sarcasm and 52% non-sarcastic observations.

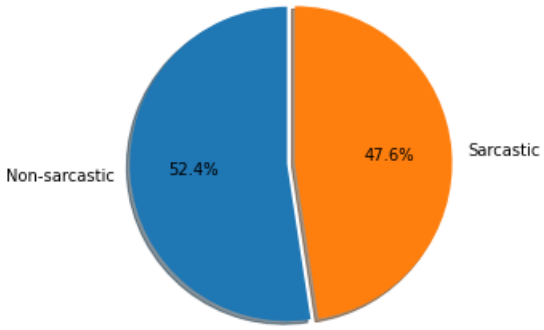

Figure 3.1: Class distribution of News Headlines Dataset

Taking a further look at the data, the word count of each observation is quite similar and nearly normally distributed in both classes as shown in figure 3.2. To have a clearer plot, word counts under 18 are plotted with 6 headlines excluded in the sarcastic class and 1 headline excluded in the non-sarcastic class.

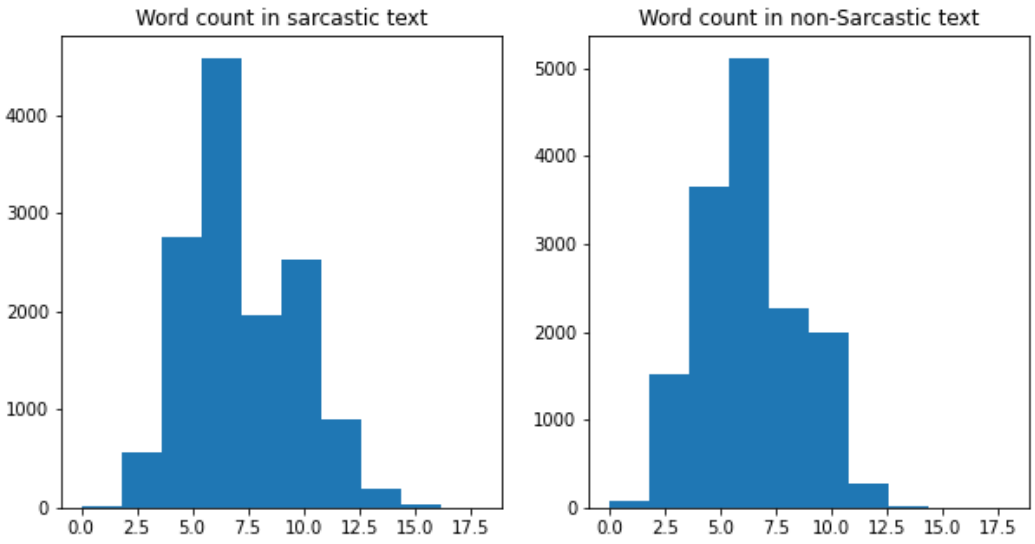

Figure 3.2: Word count distribution of sarcastic and non-sarcastic headlines

The average word length distribution in the two classes is also very similar and appears to be closely normally distributed, as shown in figure 3.3.

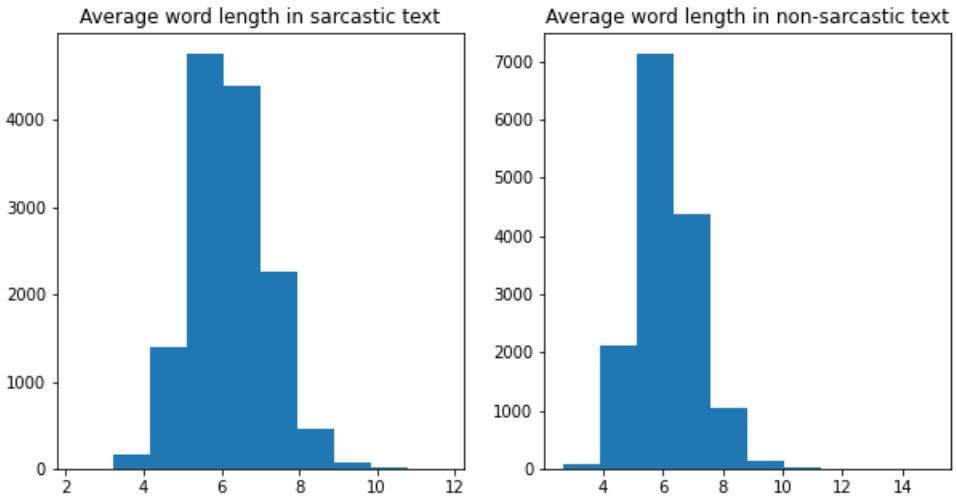

Figure 3.3: Average word length distribution of sarcastic and non-sarcastic headlines

Furthermore, the analysis of words frequency is shown as figure 3.4, with the top 50 most frequency words being plotted. It seems that both classes have their own most frequent words but also with a few words overlapped. For example, specific words like "trump" and "donald" appear to be in non-sarcastic headlines more often, whereas more general words like "man" and "new" are very frequent in both classes.

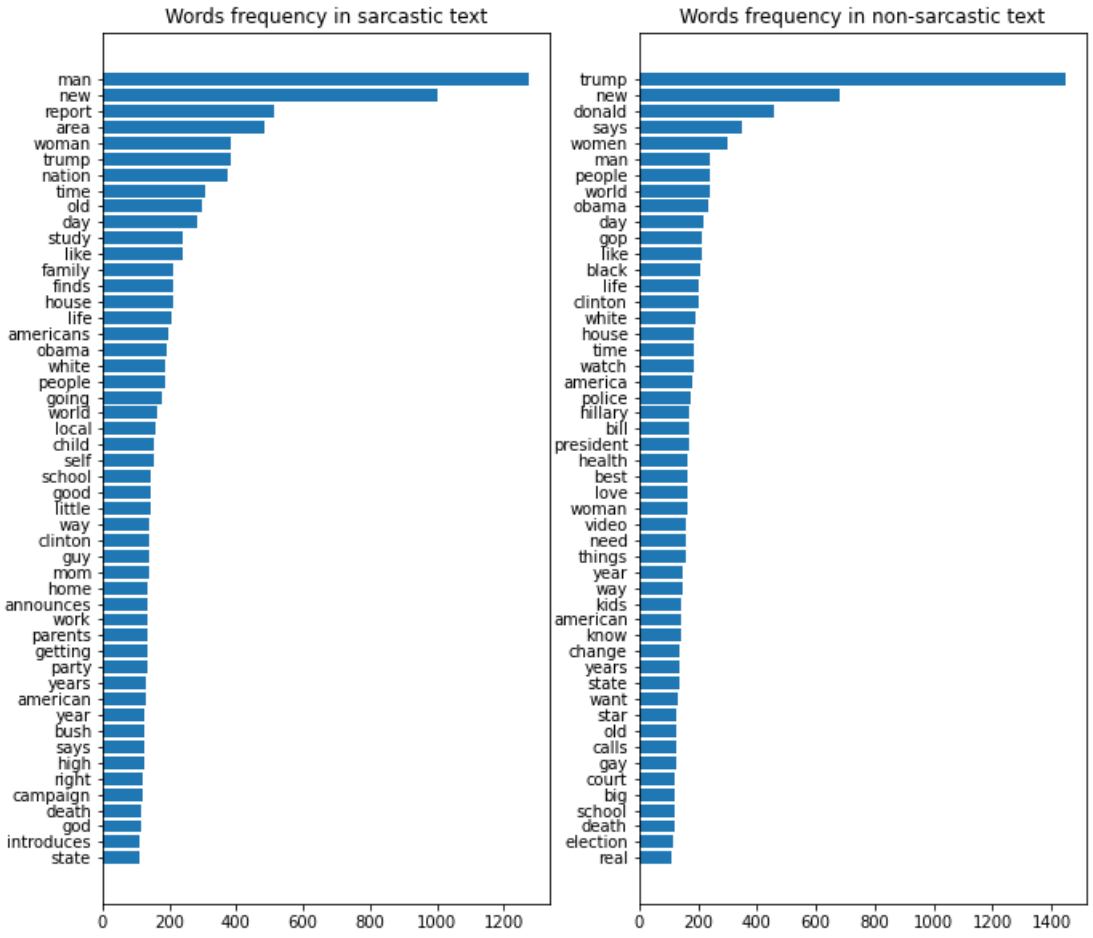

Figure 3.4: Top 50 frequent words in sarcastic and non-sarcastic headlines

From the data analysis above, it appears there is a difference between the most frequent words in different classes, although the distributions of word count and word length are more similar. It can be assumed that Naive Bayes classifier with Count Vectorizer may be able to classify and perform well up to a certain level. However, this data analysis cannot show how much the meaning of each word or the relationship between words in a context influences the classified label. This makes it interesting to see how much improvement it can get when using a more advanced neural network model with word embeddings, compared to the Naive Bayes model. Moreover, from the most frequent words figure 3.4 shows, it seems that most the words are

general and do not have a specific domain. This suggests the possibility to use pre-trained word vectors directly for the embedding matrix.

Before applying the data on models, the data is randomly shuffled to make the observations from different classes distributed more evenly throughout the split datasets. Then the data is split into training, validation, and test datasets with 70%, 15%, and 15% respectively.

4. Method

4.1 Naive Bayes Classifier[baseline]

As stated previously when analyzing the data, along with its simplicity, Naive Bayes is an appropriate method to use as the baseline for this task. A pipeline using Count Vectorizer and Multinomial Naive Bayes classifier with default settings is used on the training dataset. Then the same classifier with default settings but using Tf-idf Vectorizer is also constructed.

With the classifier and Count Vectorizer, the best set of their parameters are selected by cross-validation process using grid-search with the combination of training and validation dataset. To be more specific, the experimenting parameters include if using binary (represent the occurrence of a word as 0 and 1 instead of the number of appearances), using uni-grams, bi-grams, or both for the vectorizer, and choosing appropriate additive smoothing parameter for the multinomial classifier.

The chosen best hyper-parameters for the final Naive Bayes model are binary, both uni-grams and bi-grams, and the smoothing factor as 1.

4.2 LSTM

First, an embedding matrix is constructed using GloVe-twitter-200d pre-trained word embedding. The GloVe-twitter-200d word embedding includes 27 billion tokens, 1.2 million vocabularies, and 200-dimension in vector space, with a total size of 1.42 GB. This is chosen by its rich vocabulary over the Wikipedia word embedding and is more suitable for the computational resource at hand than the Common Crawl one which is 300-dimension. The headline dataset used in this project does not have a really rich vocabulary that can cover a large number of rare words that may come up in future unseen test data. Also because learning own embeddings from scratch requires training a huge number of parameters. Thus a word-level pre-trained word embedding - GloVe is used here. The "glove-twitter-200" word vector is downloaded and used on mapping the vocabulary in the training dataset into the embedding matrix.

The three datasets are then tokenized and pad to a maximum length of 20 for the shorter sentences, truncate till 20 words for the longer ones. The length is decided based on the distribution of the word count of the data as figure 3.2 shows.

Then an LSTM neural network model is built. After many times of experimenting and tuning on hyper-parameters using grid-search and holdout approach, the final structure of this model is decided. At the first layer - embedding layer, the constructed embedding matrix is used as weights, and the trainable parameter is set to false as previously explained. Then a bidirectional layer using LSTM unit is constructed with 128 units, 0.5 drop-out rate, and sequences is returned from this layer. Sequences are returned to make use of the following global max-pooling layer. Then followed by a dense layer with 32 units and ReLU activation function, one more drop-out layer with 0.35 drop-out rate, and a final output layer with sigmoid activation function. The number and type of layers, number of units for each layer, and the drop-out rate are adjusted based on the experimenting on the model fitting process. Drop-out rates are specifically adjusted for the over-fitting problem that appears during validating. The result RNN model with LSTM is then summarized as table 4.1.

Table 4.1: Model summary for LSTM model

Layer(type)	Output Shape	Parameters
Embedding	(None, 20, 200)	4310400
Bidirectional	(None, 20, 256)	336896
GlobalMaxPooling1D	(None, 256)	0
Dense	(None, 32)	8224
Dropout	(None, 32)	0
Dense	(None, 1)	33

The total amount of parameters that need train in this model is 345,153, with additional 4,310,400 non-trainable parameters thanks to the pre-trained GloVe word embedding.

Table 4.2: Total parameters summary for LSTM model

Name	Numbers
Total params	4,655,553
Trainable params	345,153
Non-trainable params	4,310,400

Finally, the model is compiled with loss being as binary cross-entropy, Adam optimizer with learning rate 0.001, and metrics set as accuracy. The learning rate is also tuned by looking at the changes of training and validation loss at every epoch.

Moreover, the model will be trained for 20 epochs at most and trained on mini-batches to speed up with a batch size of 128. An early stopping callback function is added to keep monitoring the validation loss during training and early stop when there is no improvement within 5 epochs to further avoid over-fitting and calculation resource wasted. Another callback function - model checkpoint is added to save the best model with the highest validation accuracy during training.

4.3 BERT

As explained, the BERT BASE model is used for this task. The construction of this model is rather straightforward. Hugging Face implementation of BERT including the transformers, pre-trained BERT BASE model and the tokenizer is used, with global average pooling on the pre-trained embedding, and one additional output layer with sigmoid activation function. The structure of the model is shown as table 4.3.

Table 4.3: Model summary for BERT model
(TFBaseModelOutputWithPoolingAndCrossAttentions)

Layer(type)	pre-trained BERT model	Output Shape	Parameters
InputLayer		(None, 20)	0
TFBertModel	last hidden state	(None, 20, 768)	
	pooler output	(None, 768)	
	past key values	None	
	hidden states	None	109482240
	attentions	None	
	cross attentions	None	
GlobalAveragePooling1D		(None, 768)	0
Dense		(None, 1)	769

The total amount of parameters that need to train in this model is around 110 million, including 109.48 million from the BERT BASE model, and additional 769 parameters that need to train from scratch for the final output layer added for this task.

Table 4.4: Total parameters summary for BERT model

Name	Numbers
Total params	109,483,009
Trainable params	109,483,009
Non-trainable params	0

The model is then compiled with Adam optimizer with learning rate 2e-5, binary cross-entropy loss method, and accuracy metric. The same early stopping and model checkpoint callback functions used in the LSTM model are added for this model as well. Finally, the model is trained with 128 batch-size and at most 10 epochs. The learning rate is decided by the suggestion of the BERT paper(Devlin et al., 2018) and experiments of the training phase.

5. Results

5.1 Naive Bayes Classifier[baseline]

The accuracy of the learned Naive Bayes model using Count Vectorizer with default settings on the test dataset is 79.37%, with the classification report shown as table 5.1. The F1 score for both classes are similar, 0.80 for non-sarcastic and 0.78 for sarcastic, with the non-sarcastic class being slightly higher. This corresponds to the observations in non-sarcastic are a bit more in the data in figure 3.1.

Table 5.1: Classification report for Naive Bayes method. With default settings and Count Vectorizer

Name	Precision	recall	F1 score	support
non-sarcastic	0.80	0.80	0.80	2241
sarcastic	0.78	0.79	0.78	2029
accuracy			0.79	4270
macro avg	0.79	0.79	0.79	4270
weighted avg	0.79	0.79	0.79	4270

The classification report for the Naive Bayes model using Tf-idf Vectorizer with default settings is shown in table 5.2, with an accuracy of 79.04%. The F1 score here has a greater difference between the two classes (0.81 for non-sarcastic and 0.77 for sarcastic), compared to the model using Count Vectorizer. The accuracy is also slightly lower.

Table 5.2: Classification report for Naive Bayes method. With default settings and Tf-idf Vectorizer

Name	Precision	recall	F1 score	support
non-sarcastic	0.79	0.83	0.81	2241
sarcastic	0.80	0.75	0.77	2029
accuracy			0.79	4270
macro avg	0.79	0.79	0.79	4270
weighted avg	0.79	0.79	0.79	4270

The model using Count Vectorizer is slightly better. As a result, the Count Vectorizer is selected to proceed and run through the cross-validation process using Grid Search to select the best hyper-parameters. The resulted selected parameters are **binary** being as **True**, **ngram_range** being as **(1,2)**, and **alpha** being as **1**. The classification report for this fine-tuned model is shown in table 5.3. The F1 scores for both classes and the accuracy for this model are all improved by the optimized hyper-parameters. The F1 score for the non-sarcastic class is improved from 0.80 to 0.81, similarly with the sarcastic class (increased from 0.78 to 0.79). The test accuracy is improved slightly to 80%.

Table 5.3: Classification report for Naive Bayes method. With optimized settings and Count Vectorizer

Name	Precision	recall	F1 score	support
non-sarcastic	0.81	0.81	0.81	2241
sarcastic	0.79	0.79	0.79	2029
accuracy			0.80	4270
macro avg	0.80	0.80	0.80	4270
weighted avg	0.80	0.80	0.80	4270

The corresponding confusion matrix is as figure 5.1 shows. The miss-classified observations are distributed across these two classes quite evenly. 432 out of 2241 non-sarcastic cases are miss classified as sarcastic, and 422 out of 2029 sarcastic cases are wrongly identified as non-sarcastic. It further verifies by the F1 scores in the classification report in table 5.3.

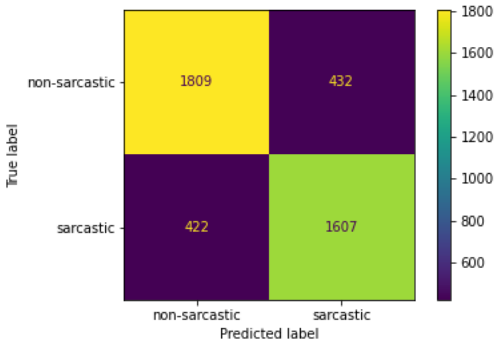

Figure 5.1: Confusion matrix of the performance of Naive Bayes model

5.2 LSTM

For the RNN model with LSTM, the accuracy and loss for training and validation datasets during the training process can be seen in figure 5.2. For a total of 20 epochs, the model has been trained for 17 epochs before it got interrupted by early stopping (with patience being as 5 epochs). The minimum loss for validation data is gained at epoch 12, then it increases while the training data loss keeps going down, which suggests a potential over-fitting for the later-on training process. This further proved that early stopping is required here. The best model is saved at epoch 14 with the maximum validation accuracy.

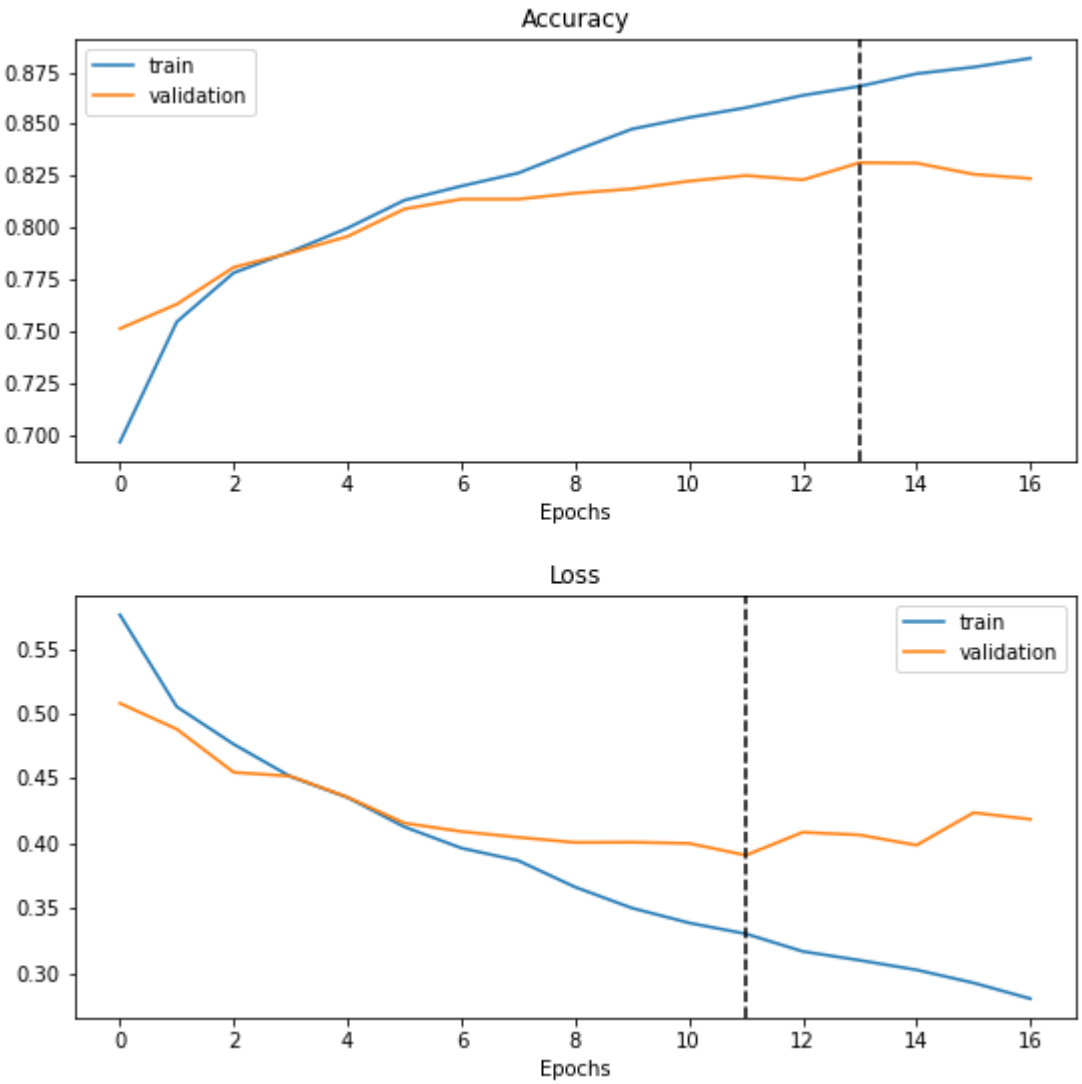

Figure 5.2: Accuracy and loss changing during training for LSTM

The accuracy on the test dataset is 82.71%, with the confusion matrix shown in figure 5.3. The miss-classified rate is very similar in the two classes, with 350 out of 2241 sarcastic cases being miss classified as non-sarcastic, and 388 out of 2029 non-sarcastic cases being incorrectly identified as sarcastic.

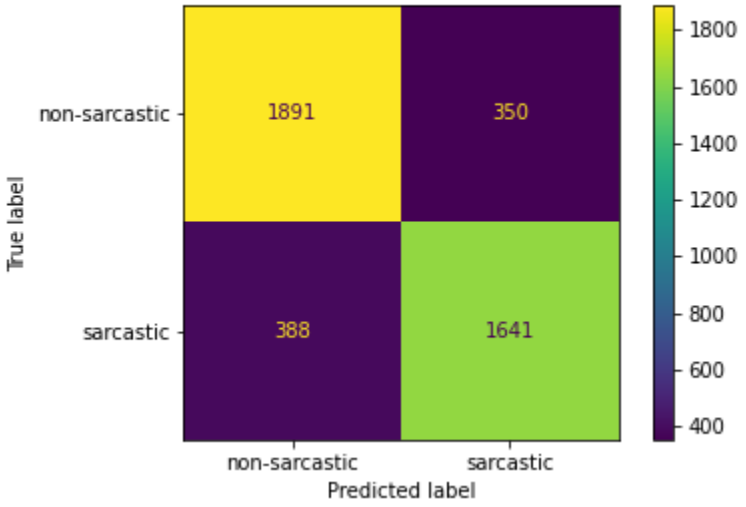

Figure 5.3: Confusion matrix of the performance of LSTM model

The F1 score and accuracy that corresponds to this model is as shown in table 5.4. As the confusion matrix showed, the F1 scores for both classes are alike, with 0.83 for non-sarcastic and 0.82 for sarcastic. The difference in precision between the two classes is a bit big, with 0.84 for non-sarcastic and 0.81 for sarcastic.

Table 5.4: Classification report for LSTM model

Name	Precision	recall	F1 score	support
non-sarcastic	0.83	0.84	0.84	2241
sarcastic	0.82	0.81	0.82	2029
accuracy			0.83	4270
macro avg	0.83	0.83	0.83	4270
weighted avg	0.83	0.83	0.83	4270

5.3 BERT

For the BERT model, the plot of its training process is shown in figure 5.4. The model has been trained for 5 out of 10 epochs before it was early-stopped (patience equals 3). The loss plot shows that the lowest loss for the validation dataset was gained very early (at the second epoch). The best model with the largest validation accuracy was saved at epoch 3 according to the accuracy plot.

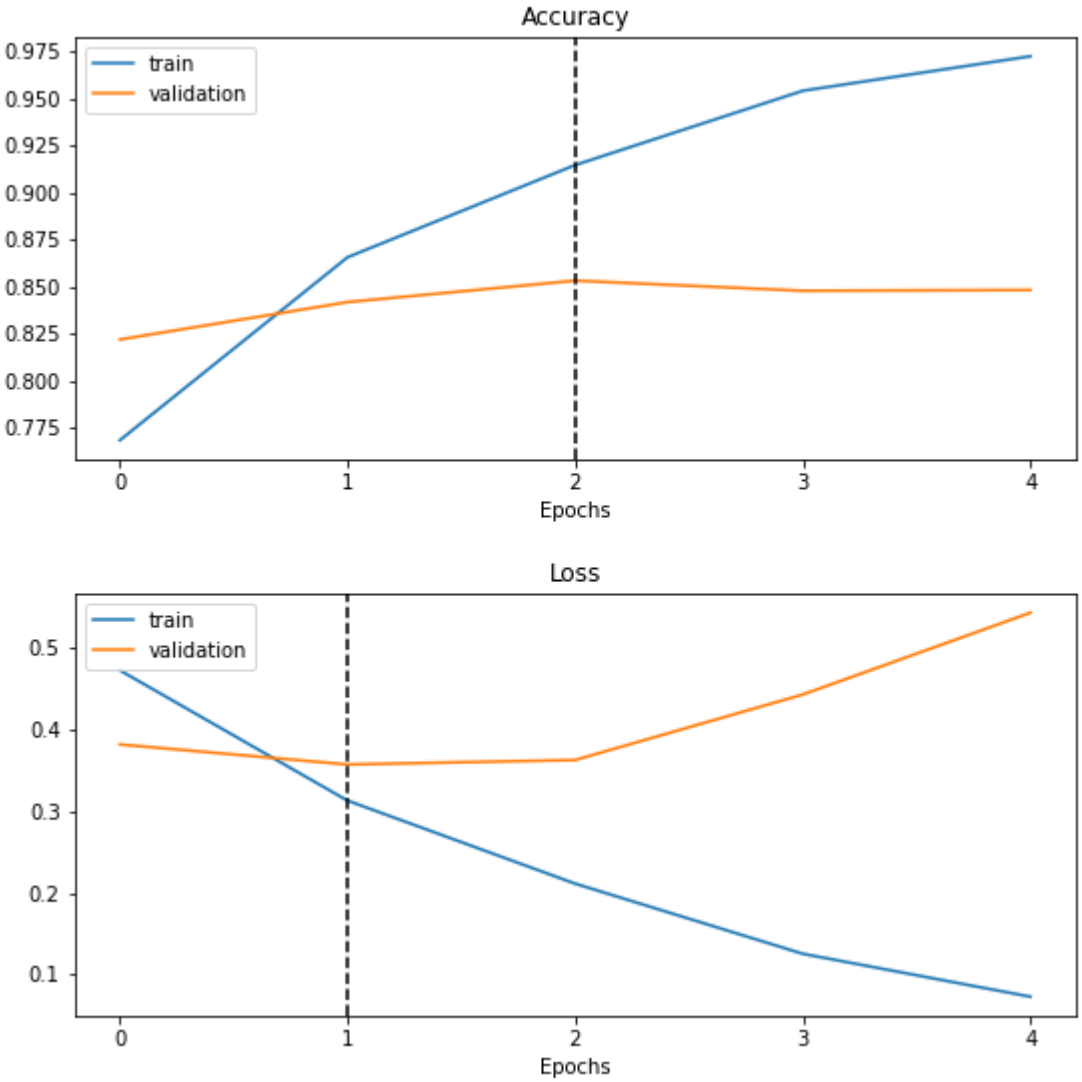

Figure 5.4: Accuracy and loss changing during training for BERT

The accuracy of this model on the test dataset is 85.88%, with the confusion matrix shown in figure 5.5. With 318 out of 2029 sarcastic cases being wrongly classified, and only 285 out of 2241 non-sarcastic being miss classified, the difference between the miss-classification rate for the two classes is small but noticeable.

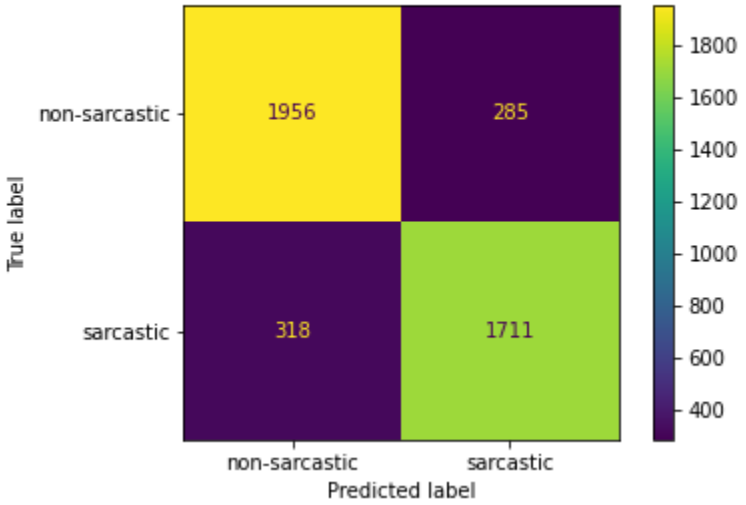

Figure 5.5: Confusion matrix of the performance of BERT model

The classification report that corresponds to this model is shown as table 5.5. As the confusion matrix shows, the recall rates for the two classes are a bit different, with 0.87 for non-sarcastic and only 0.84 for sarcastic. With the same precision rate, it results in a slightly different F1 score (0.87 for non-sarcastic and 0.85 for sarcastic). The overall accuracy for this model is 0.86.

Table 5.5: Classification report for BERT model

Name	Precision	recall	F1 score	support
non-sarcastic	0.86	0.87	0.87	2241
sarcastic	0.86	0.84	0.85	2029
accuracy			0.86	4270
macro avg	0.86	0.86	0.86	4270
weighted avg	0.86	0.86	0.86	4270

6. Discussion

6.1 Analyse results

Three models including the baseline model have been constructed, fine-tuned, and trained. All the models have shown a relatively balanced performance across the two classes in terms of precision and recall. As figure 6.1 and figure 6.2 show, the BERT model outperforms LSTM and the Naive Bayes model on both F1 score and Accuracy. The Naive Bayes model sets a high bar for the baseline. Given the restricted data, it is expected that there is a limited improvement that can be achieved even by an advanced deep learning model. Because when the model is complicated enough, it is more likely to encounter an over-fitting problem if the data at hand is not enormous.

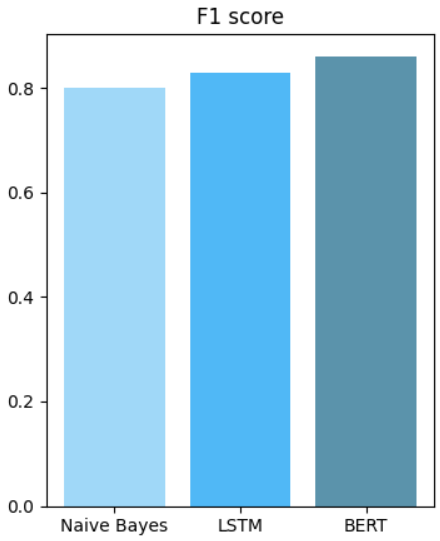

Figure 6.1: F1 score across all models

In this project, a pre-trained BERT model has been transferred onto this specific task and fine-tuned all the parameters. Considering the complexity of this pre-trained model, the relatively simple data, and this rather small task (classification with two classes), the construction of the model becomes very straightforward, and so does the tweaking of its hyper-parameters. The training process takes a lot longer time per epoch than LSTM (around 43s vs. 2s on the device used for this project), but it converges quite early, always within 5 epochs during experimenting. With only one additional output layer, the BERT model still shows an over-fitting trend (figure 5.4). With the help of early stopping and loss monitoring, a model with acceptable balancing between generalization and over-fitting is obtained. As a result, the BERT model is approximately 6 percent higher than the baseline model in the F1 score. And it is 3 percent higher than LSTM. A similar

situation happens with the test accuracy metric (BERT 86% vs. LSTM 83%).

The LSTM model is 3 percent better on both F1 score and accuracy than the Naive Bayes model (LSTM 83% vs. NB 80%). The model is built from the ground. Thus, there are a lot more to consider, for example, the types of the layers, the structure of the layers, and the hyper-parameters within the model. The construction and hyper-parameters tweaking phase take more effort than the BERT model. The training speed for this model is fairly fast even with the 300-dimensional word vectors. It is mainly because the length of the data is rather small (20 words max), and it is also perhaps slightly influenced by the max-pooling layer used here. From the training phase, an over-fitting trend can also be seen in figure 5.2. The gap between validation loss and training loss does not increase as rapidly as the BERT model by epoch. From figure 5.2, the final model is selected by considering the trade-off of variance and bias. Overall, it appears that the BERT model outperforms the LSTM model, and achieves a clear improvement compared to the baseline.

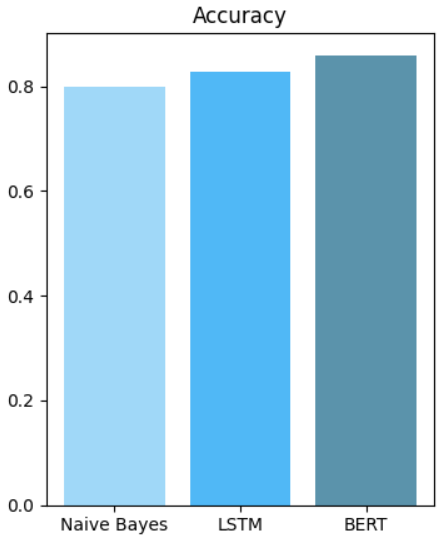

Figure 6.2: Accuracy across all models

In addition, some wrongly classified cases are filtered and presented in table 6.1. The reason that the first two cases in the table are wrongly classified may be that "trump", "hillary", and "clinton" are among the top frequent words in the non-sarcastic class, which can be seen in figure 3.4. The algorithm then tends to put more weight on the sentences containing these words to classify them as non-sarcastic. With more time and data, this problem can be addressed by having equal training data at the same period. Because news tends to talk about the same events that happen during the same period, which is more likely to provide topic balanced data. For example, both sarcastic and non-sarcastic websites release similar amounts of articles with headlines containing "trump" or "clinton", then those words will not be heavily distributed towards only one class.

The pre-processing procedure sometimes causes losing information and even reversing meaning. The third example shows that the pre-processing removed the "-" and stop word "that", which causes losing information

("that" means the sub-sentence before "-"). In the second example, "by" is removed and the sentence becomes "hillary clinton threatened black man", while it really should be "hillary clinton threatened **by** black man". Moreover, in the fourth example, "having squad makes life **less** painful" becomes to "having squad makes life painful". Losing "less" reversed the original meaning. In the last example, the rhetorical question style was completely broken. For the sentences that were properly processed, the models are not likely to predict them well. A more complicated pre-process has to be researched to deal with this problem.

Table 6.1: Wrongly classified cases by both LSTM and BERT

Headline	After pre-process	true label
trump motorcade picks up few lyft passengers to help president make ends meet	trump motorcade picks lyft passengers help president ends meet	1
hillary clinton threatened by black man	hillary clinton threatened black man	1
i'm a nightmare ex-girlfriend — and i'm cool with that	nightmare ex girlfriend cool	0
scientific proof that having a squad makes life less painful	scientific proof having squad makes life painful	0
mass graves: are they really more cost-effective?	mass graves cost effective	1

6.2 Related work

Sarcasm recognition has drawn much attention from researchers. Much related work has been done and many models have been purposed. Among those, hybrid models using pre-trained word embedding and later performing neural network classification using LSTM and CNN have shown a competitive performance. Mehndiratta and Soni (2019) analyzed the performance of multiple hybrid frameworks on the detection of sarcasm. They evaluated the models on three different datasets, including a news headlines dataset (which is very similar to the dataset in this project). In the article, they concluded that the hybrid model with LSTM-CNN structure (shown in figure6.3 (Mehndiratta and Soni,2019)) performs the best on all their datasets.

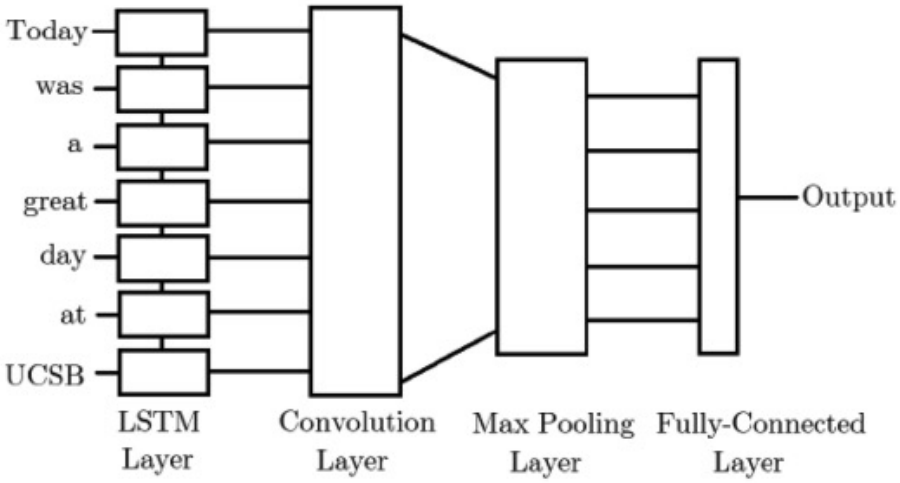

Figure 6.3: Compared framework: LSTM-CNN architecture

In this project, the same LSTM-CNN architecture is constructed and fine-tuned on the same dataset using in this project. The model uses the same GloVe word embedding as the LSTM model. The final model is shown as in table 6.2. The main difference between this hybrid model and the LSTM model appears to be the additional convolution layer.

Table 6.2: Model summary for hybrid model

Layer(type)	Output Shape	Parameters
Embedding	(None, 20, 200)	4310400
Bidirectional	(None, 20, 256)	336896
Conv1D	(None, 18, 64)	49216
MaxPooling1D	(None, 9, 64)	0
Dropout	(None, 9, 64)	0
Flatten	(None, 576)	0
Dense	(None, 1)	577

The word vectors are also be set as not trainable as the LSTM model does. The total trainable parameter in this hybrid model is 386,689, which is around 40,000 more than the LSTM model without the CNN layer.

Table 6.3: Total parameters summary for hybrid model

Name	Numbers
Total params	4,697,089
Trainable params	386,689
Non-trainable params	4,310,400

Similar to the LSTM and BERT models, the hybrid model also shows a clear over-fitting trend. The hybrid model converges faster than LSTM, but slower than BERT. It only takes only 6-7 epochs on average, compare to 12-14 epochs for LSTM, and 2-3 for BERT. The accuracy on test data for the hybrid model is 81.31%, which is slightly lower than the LSTM model and much lower than BERT. Considering the randomness, it can be concluded that the LSTM model and hybrid model are similar in accuracy performance.

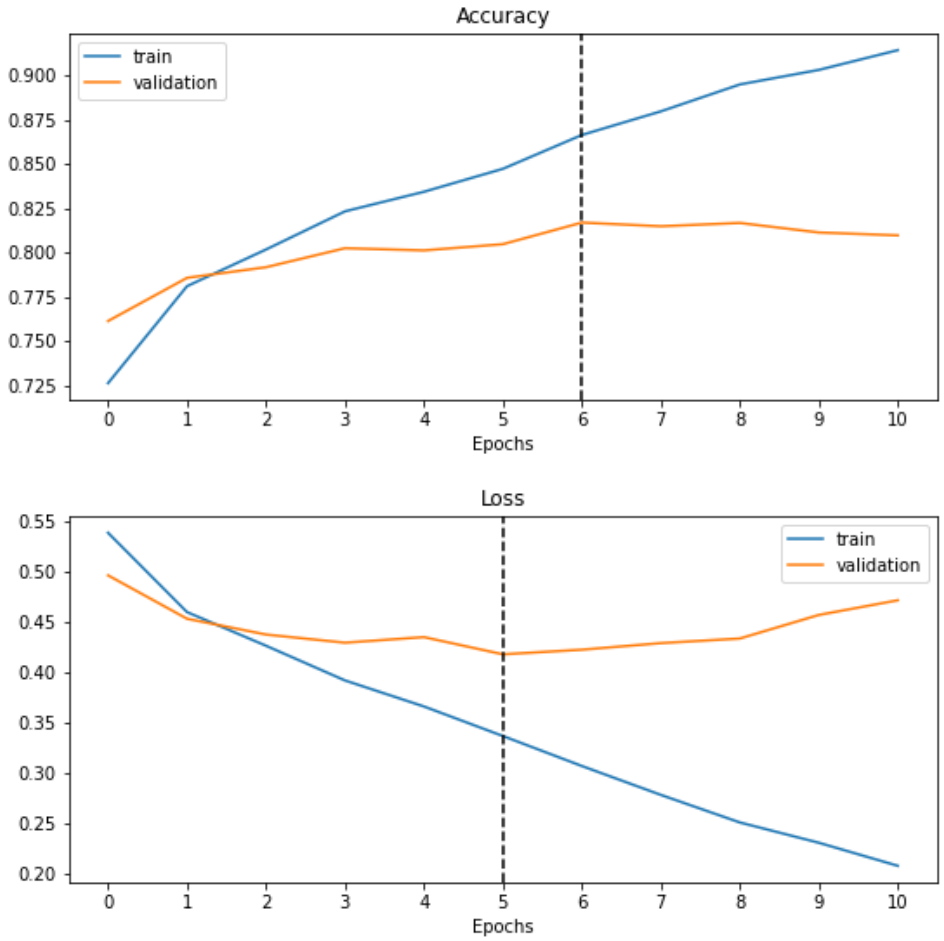

Figure 6.4: Accuracy and loss changing during training for hybrid model

However, the classification report shows a rather unbalanced result in terms of F1 score. The precision of non-sarcastic classification is much higher than the sarcastic classification(0.86 vs. 0.77). While for recall, the sarcastic class is greatly higher than non-sarcastic (also with 0.86 vs. 0.77). This indicates that the hybrid model is better at classifying sarcastic than non-sarcastic observations, which can be further proved in confusion matrix in figure 6.5.

Table 6.4: Classification report for hybrid model

Name	Precision	recall	F1 score	support
non-sarcastic	0.86	0.77	0.81	2241
sarcastic	0.77	0.86	0.81	2029
accuracy			0.81	4270
macro avg	0.82	0.82	0.81	4270
weighted avg	0.82	0.81	0.81	4270

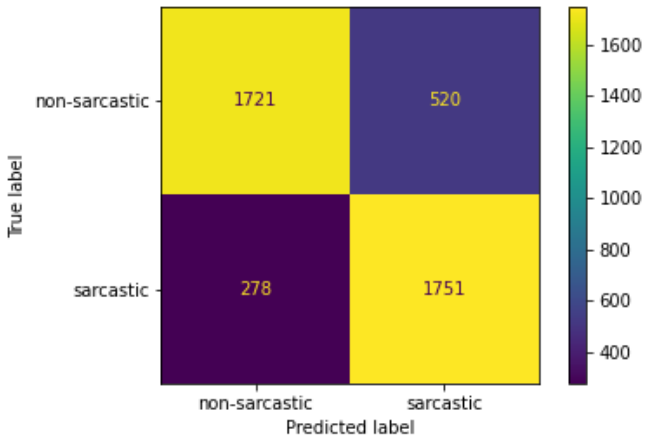

Figure 6.5: Confusion matrix of the performance of hybrid model

The hybrid model may have a potentially better performance on larger scale sequence data. Because the CNN layer is good at highlighting the related information. It may also show a clearer advantage in terms of training speed on larger dataset due to CNN. In this project, it is already over-fitting for the simpler LSTM model. It is expected that the more complex hybrid model would overfit the data or overfit one class than the other.

7. Conclusion

The fact that social media has become more and more essential to our daily fast-paced lives has led to an online environment with overwhelming amounts of information. Much of this information can be misleading. One instance of misleading information is news sites using sarcasm to attract readers with outrageous headlines. To avoid people wrongly believing such information to be true, it would be useful to have the ability to automatically identify sarcastic headlines. In this project, LSTM, BERT, and Naive Bayes baseline models are presented. Both LSTM and BERT show a better result than the baseline model. With more time and computational resources, a better architecture may be found for the LSTM model. A hybrid model from a related work is also reproduced and compared. The BERT model performs the best in terms of the difficulty of deployment and performance.

In terms of data, one more approach that could be considered is to make use of the article-link column. It can be helpful to predict with the help of the actual full article text. Moreover, the date when the news is released may also be an important factor to make the data more topic-balanced throughout classes, as previously explained. However, detecting sarcasm is far beyond only classical text analysis. In a paper(Ivanko & Pexman, 2003), they indicated that sarcasm usually consists of Speaker, Listener, Context, Utterance, Literal, and Intended Prepositions. The sentence style (i.e. rhetorical question) of expressing can also be critical to determine whether it is sarcastic or not. As the wrongly classified cases table 6.1 shows, the classic pre-process procedure has a clear drawback on better presenting the data. Thus, this project can also be further extended to come up with a more dynamic pre-processing method that is better suitable for sarcastic text.

References

Barbieri, F., Saggion, H., & Ronzano, F. (2014). Modelling sarcasm in twitter, a novel approach. In Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, 50–58.

Deerwester, Scott., T, Susan., Dumais, George W., Furnas, Thomas K., Landauer., & Richard Harshman. (1990). Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41.

Devlin, Jacob., Chang, Ming-Wei., Lee, Kenton., Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805v2.

Goldberg, Yoav. (2017). Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies), 92.

Hastie, Trevor. (2001). The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations. Tibshirani, Robert., Friedman, J. H. (Jerome H.). New York: Springer.

Hochreiter, Sepp., Schmidhuber, Jürgen. (1997). Long short-term memory. Neural Computation. `https://doi.org/10.1162/neco.1997.9.8.1735`

McCallum, Andrew., & Nigam, Kamal. (1998). A comparison of event models for Naive Bayes text classification (PDF). AAAI-98 workshop on learning for text categorization. 752.

Mikolov, Tomas., Chen, Kai., Corrado, Greg., & Dean Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. In ICLR Workshop Papers.

Misra, Rishabh. (2021). News headlines dataset for sarcasm detection: High quality dataset for the task of sarcasm detection. `https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection`

Pennington, Jeffrey., Socher, Richard, & Manning, Christopher. (2014). *GloVe: Global Vectors for Word Representation*(Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing). Association for Computational Linguistics. `https://aclanthology.org/D14-1162`

Vaswani, Ashish., Shazeer, Noam., Parmar, Niki., Uszkoreit, Jakob., Jones, Llion., Gomez, Aidan N.., Kaiser, Lukasz., & Polosukhin, Illia. (2017). Attention Is All You Need. arXiv:1706.03762.

Zhu, Yukun., Kiros, Ryan., Zemel, Rich., Salakhutdinov, Ruslan., Urtasun, Raquel., Torralba, Antonio., & Fidler, Sanja. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In Proceedings of the IEEE international conference on computer vision, 19–27.